

图文 044、高级工程师的硬核技能：JVM的Young GC日志应该怎么看

1924 人次阅读 2019-08-13 07:00:00

详情 评论

高级工程师的硬核技能：

JVM的Young GC日志应该怎么看？

给大家推荐一套质量极高的Java面试训练营课程：



作者是中华石杉，石杉老哥是我之前所在团队的 Leader，骨灰级的技术神牛！

大家可以点击下方链接，了解更多详情，并进行试听：

[21天互联网Java进阶面试训练营（分布式篇）](#)

1、前文回顾

昨天的文章我们给大家了一个示例性的代码，同时给出了实验用的JVM参数，教会了大家如何在Eclipse中去使用指定的JVM参数运行程序，并且看到了代码运行触发的Young GC的日志，然后先让大家自己尝试思考和分析一下。

今天我们的文章就通过一步一图的方式，接着昨天的内容继续来仔细分析GC日志，同时配合画图告诉大家依次Young GC的实际执行过程。

2、程序运行采用的默认JVM参数如何查看？

在GC日志中，可以看到如下内容：

```
CommandLine flags: -XX:InitialHeapSize=10485760 -XX:MaxHeapSize=10485760 -XX:MaxNewSize=5242880 .....
```

这就是告诉你这次运行程序采取的JVM参数是什么，基本都是我们设置的，同时还有一些参数默认就给设置了，不过一般关系不大。

告诉大家这个，是很多人问我，如果没设置JVM参数的话，怎么看系统用的默认JVM参数？

很简单，给你的JVM起码加一段打印gc日志的参数，就可以在这里看到他默认会给你的JVM进程分配多大的内存空间了。默认给的内存是很小的。

3、一次GC的概要说明

接着我们看GC日志中的如下一行：

```
0.268: [GC (Allocation Failure) 0.269: [ParNew: 4030K->512K(4608K), 0.0015734 secs] 4030K->574K(9728K), 0.0017518 secs]
[Times: user=0.00 sys=0.00, real=0.00 secs]
```

这个就是概要说明了本次GC的执行情况，给大家讲一遍，大家就知道怎么回事了。

GC (Allocation Failure)，这个看字面意思就知道，为啥会发生一次GC呢？

很简单，因为看上图，要分配一个2MB的数组，结果Eden区内存不够了，所以就出现了“Allocation Failure”，也就是对象分配失败。

所以此时就要触发一次Young GC。

那这次GC是什么时候发生的呢？

很简单，看一个数字，“0.268”，这个意思就是你的系统运行以后过了多少秒发生了本次GC，比如这里就是大概系统运行之后过了大概200多毫秒，发生了本次GC。

```
ParNew: 4030K->512K(4608K), 0.0015734 secs
```

这个“ParNew”的意思，大家很明确了吧，我们触发的是年轻代的Young GC，所以是用我们指定的ParNew垃圾回收器执行GC的。

4030K->512K(4608K)

这个代表的意思是，年轻代可用空间是4608KB，也就是4.5MB，为啥是4.5MB呢？

大家看上图，Eden区是4MB，两个Survivor中只有一个是放存活对象的，另外一个必须保持一致空闲的，所以他考虑年轻代的可用空间，就是Eden+1个Survivor的大小，也就是4.5MB。

然后4030K->512K，意思就是对年轻代执行了一次GC，GC之前都使用了4030KB了，但是GC之后只有512KB的对象是存活下来的。

0.0015734 secs，这个就是本次gc耗费的时间，看这里来说大概耗费了1.5ms，仅仅是回收3MB的对象而已。

4030K->574K(9728K), 0.0017518 secs，这段话指的是整个Java堆内存的情况

意思是整个Java堆内存是总可用空间9728KB（9.5MB），其实就是年轻代4.5MB+老年代5M，然后GC前整个Java堆内存里使用了4030KB，GC之后Java堆内存使用了574KB。

[Times: user=0.00 sys=0.00, real=0.00 secs]

这个意思就是本次gc消耗的时间，大家可以看，这里最小单位是小数点之后两位，但是这里全部是0.00 secs，也就是说本次gc就耗费了几毫秒，所以从秒为单位来看，几乎是0。

相信大家看到这里，会有很多疑惑，感觉好像跟自己的认知有一些差距

不要紧，稍安勿躁，继续跟着看下去，接下来我们给大家图解一下这个GC发生的过程。

4、图解GC执行过程

第一个问题，看这行日志，ParNew: 4030K->512K(4608K), 0.0015734 secs

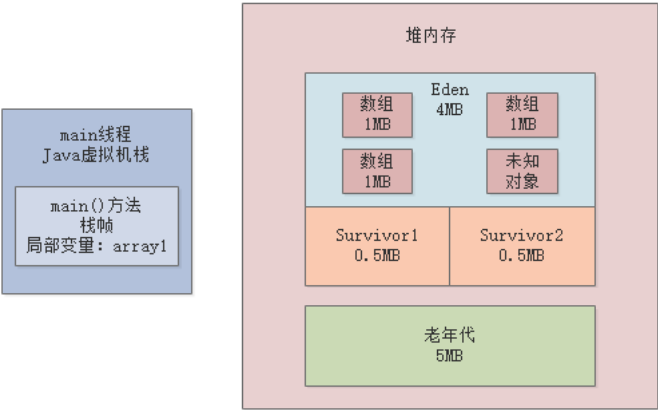
大家很奇怪，我们在GC之前，明明在Eden区里就放了3个1MB的数组，一共是3MB，也就是3072KB的对象，那么GC之前年轻代应该是使用了3072KB的内存啊，为啥是使用了4030KB的内存呢？

其实这个问题，大家先不要纠结，你只要明白两点：

其实你创建的数组本身虽然是1MB，但是为了存储这个数组，JVM内置还会附带一些其他信息，所以每个数组实际占用的内存是大于1MB的；

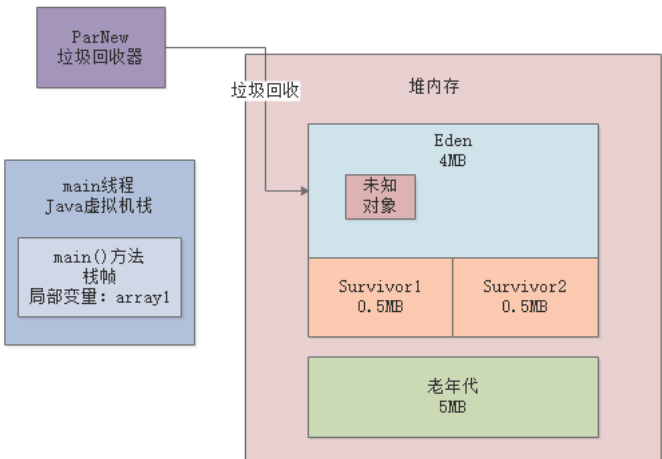
除了你自己创建的对象以外，可能还有一些你看不见的对象在Eden区里，至于这些看不见的未知对象是什么，后面我们有专门的工具可以分析堆内存快照，以后会带你看到这些对象是什么。

所以如下图所示，GC之前，三个数组和其他一些未知对象加起来，就是占据了4030KB的内存。



接着你想要在Eden分配一个2MB的数组，此时肯定触发了 “Allocation Failure ”，对象分配失败，就触发了Young GC

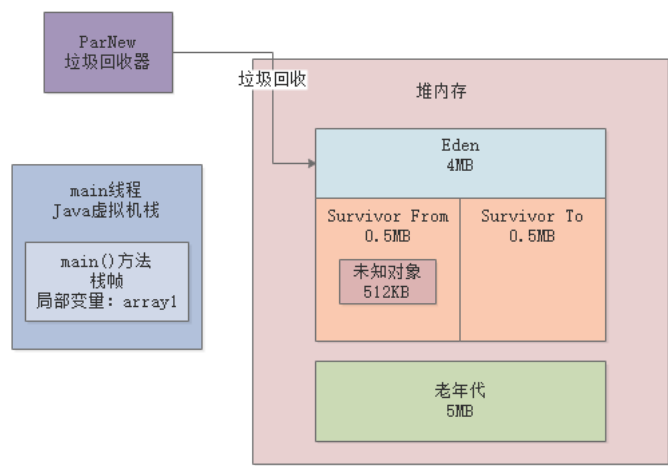
然后ParNew执行垃圾回收，回收掉之前我们创建的三个数组，此时因为他们都没人引用了，一定是垃圾对象，如下图所示。



然后我们继续看gc日志， **ParNew: 4030K->512K(4608K), 0.0015734 secs**

gc回收之后，从4030KB内存使用降低到了512KB的内存使用

也就是说这次gc日志有512KB的对象存活了下来，从Eden区转移到了Survivor1区，其实我们可以把称呼改改，叫做Survivor From区，另外一个Survivor叫做Survivor To区，如下图。



其实结合GC日志就能看出来，这就是本次GC的全过程。

5、GC过后的堆内存使用情况

接着我们看下面的GC日志：

```
Heap

par new generation  total 4608K, used 2601K [0x00000000ff600000, 0x00000000ffb00000, 0x00000000ffb00000)

eden space 4096K,  51% used [0x00000000ff600000, 0x00000000ff80a558, 0x00000000ffa00000)

from space 512K, 100% used [0x00000000ffa80000, 0x00000000ffb00000, 0x00000000ffb00000)

to   space 512K,   0% used [0x00000000ffa00000, 0x00000000ffa00000, 0x00000000ffa80000)

concurrent mark-sweep generation total 5120K, used 62K [0x00000000ffb00000, 0x0000000100000000, 0x0000000100000000)

Metaspace    used 2782K, capacity 4486K, committed 4864K, reserved 1056768K

class space   used 300K, capacity 386K, committed 512K, reserved 1048576K
```

这段日志是在JVM退出的时候打印出来的当前堆内存的使用情况，其实也很简单，一点点看一下，先看下面这段。

```
par new generation  total 4608K, used 2601K [0x00000000ff600000, 0x00000000ffb00000, 0x00000000ffb00000)
```

eden space 4096K, 51% used [0x00000000ff600000, 0x00000000ff80a558, 0x00000000ffa00000)
from space 512K, 100% used [0x00000000ffa80000, 0x00000000ffb00000, 0x00000000ffb00000)
to space 512K, 0% used [0x00000000ffa00000, 0x00000000ffa00000, 0x00000000ffa80000)

par new generation total 4608K, used 2601K, 这就是说 “ParNew” 垃圾回收器负责的年轻代总共有 4608KB (4.5MB) 可用内存，目前是使用了2601KB (2.5MB) 。

那么大家思考一下，此时在JVM退出之前，为什么年轻代占用了2.5MB的内存？

很简单，在gc之后，我们这不是通过如下代码又分配了一个2MB的数组吗：byte[] array2 = new byte[2 * 1024 * 1024];

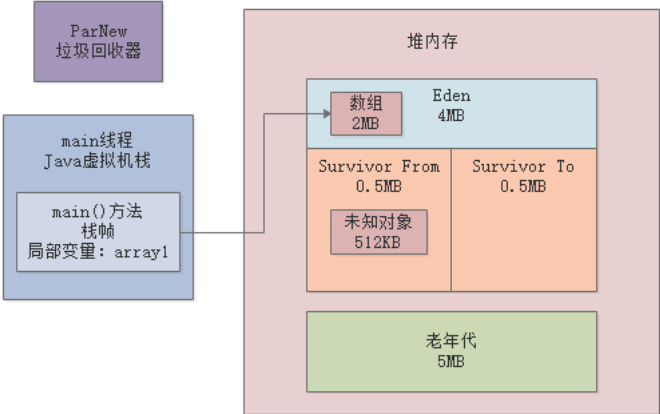
所以此时在Eden区中一定会有一个2MB的数组，也就是2048KB，然后上次gc之后在From Survivor区中存活了一个 512KB的对象，大家也不知道是啥，先不用管他。

但是此时你疑惑了，2048KB + 512KB = 2560KB。

那为什么说年轻代使用了2601KB呢？

因为之前说过了每个数组他会额外占据一些内存来存放一些自己这个对象的元数据，所以你可以认为多出来的41KB可以是数组对象额外使用的内存空间。

如下图所示。



接着我们继续看GC日志：

```
eden space 4096K, 51% used [0x00000000ff600000, 0x00000000ff80a558, 0x00000000ffa00000)
from space 512K, 100% used [0x00000000ffa80000, 0x00000000ffb00000, 0x00000000ffb00000)
to   space 512K, 0% used [0x00000000ffa00000, 0x00000000ffa00000, 0x00000000ffa80000)
```

通过GC日志就能验证我们的推测是完全准确的，这里说的很清晰了，Eden区此时4MB的内存被使用了51%，就是因为有一个2MB的数组在里面。

然后From Survivor区，512KB是100%的使用率，此时被之前gc后存活下来的512KB的未知对象给占据了。

接着看GC日志：

```
concurrent mark-sweep generation total 5120K, used 62K [0x00000000ffb00000, 0x0000000100000000,
0x0000000100000000)

Metaspace    used 2782K, capacity 4486K, committed 4864K, reserved 1056768K

class space  used 300K, capacity 386K, committed 512K, reserved 1048576K
```

concurrent mark-sweep generation total 5120K, used 62K，这个很简单，就是说Concurrent Mark-Sweep垃圾回收器，也就是CMS垃圾回收器，管理的老年代内存空间一共是5MB，此时使用了62KB的空间，这个是啥你也先不用管了，可以先忽略不计，以后我们有内存分析工具了，你都能看到。

```
Metaspace    used 2782K, capacity 4486K, committed 4864K, reserved 1056768K

class space  used 300K, capacity 386K, committed 512K, reserved 1048576K
```

上述两段日志也很简单，意思就是Metaspace元数据空间和Class空间，存放一些类信息、常量池之类的东西，此时他们的总容量，使用内存，等等。

6、今日思考题

昨天的思考题其实在今天的文章里就解答了，但是今天要给大家留一个小的思考题。

```
Metaspace    used 2782K, capacity 4486K, committed 4864K, reserved 1056768K

class space  used 300K, capacity 386K, committed 512K, reserved 1048576K
```

对JDK 1.8以后的Metaspace和Classspace，大家去百度一下，这里都是存放什么内容的

然后gc日志中这里的used、capacity、committed、reserved几个字段，都表示什么含义？希望大家自己去探索一下。


结合专栏的学习，同时自己探索一些小细节，是一件很有意思的事情，大家加油，有心得可以在评论区发出来，给其他人分享一下。

End

专栏版权归公众号**狸猫技术窝**所有

未经许可不得传播，如有侵权将追究法律责任

Copyright © 2015-2019 深圳小鹅网络技术有限公司 All Rights Reserved. 粤ICP备15020529号

 小鹅通提供技术支持